

An Effective Procedure for Computing “Uncomputable” Functions*

Kurt Ammon[†]

Abstract

We give an effective procedure that produces a natural number in its output from any natural number in its input, that is, it computes a total function. The elementary operations of the procedure are Turing-computable. The procedure has a second input which can contain the Gödel number of any Turing-computable total function whose range is a subset of the set of the Gödel numbers of all Turing-computable total functions. We prove that the second input cannot be set to the Gödel number of any Turing-computable function that computes the output from any natural number in its first input. In this sense, there is no Turing program that computes the output from its first input. The procedure is used to define creative procedures which compute functions that are not Turing-computable. We argue that creative procedures model an aspect of reasoning that cannot be modeled by Turing machines.

1 Introduction

There are doubts whether Turing machines can capture all reasoning processes. For example, Turing [1939, pp. 200, 215] writes:

Gödel’s theorem shows that such a system [intellectually satisfying system of logical inference] cannot be wholly mechanical ...

*This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License (see <http://creativecommons.org/licenses/by-nc-nd/3.0/>).

[†]Correspondence to p.k.ammon [at] cproc [period] org. Comments are welcome.

The necessity for using the intuition is then [by the introduction of a formal logic] greatly reduced by setting down formal rules for carrying out inferences which are intuitively valid. ... In pre-Gödel times it was thought by some that it would probably be possible to carry this programme to such a point that all the intuitive judgments of mathematics could be replaced by a finite number of these rules. The necessity for intuition would then be entirely eliminated.

Thus, Turing [1939] interprets Gödel's theorem in the sense that an "intellectually satisfying system of logical inference ... cannot be wholly mechanical" and "all the intuitive judgments of mathematics" cannot "be replaced by a finite number of these rules", that is, "formal rules for carrying out inferences which are intuitively valid".

Turing [1936, p. 231] restricts his machines to a finite number of *m*-configurations (machine configurations) which are called "states of mind" in his [1936, pp. 249-250] substantiation of the thesis that his machines can compute "all numbers which would naturally be regarded as computable" (Turing's thesis). Turing [1936, pp. 249-250] supposes that the "number of states of mind" is finite because some of them "will be confused" if "we admitted an infinity of states of mind" (see Kleene [1952, pp. 376-377]). Gödel [1990, p. 306] regards the restriction to a finite number of states as a "philosophical error in Turing's work" and points out that "mental procedures" may "go beyond mechanical procedures". Gödel [1990, p. 306] writes:

What Turing disregards completely is the fact that *mind, in its use, is not static, but constantly developing*, ... There may exist systematic methods of actualizing this development, which could form part of the procedure. Therefore, although at each stage the number and precision of the abstract terms at our disposal may be *finite*, both (and, therefore, also Turing's number of *distinguishable states of mind*) may *converge toward infinity* in the course of the application of the procedure.

Thus, Gödel discusses the possible existence of "mental" procedures that cannot be modeled by any Turing machine which is restricted to a finite number of "states of mind" in Turing's [1936, pp. 249-250] substantiation of his thesis.

Referring to his own form of Gödel's incompleteness theorem, Post [1944, p. 295] writes:

The conclusion is unescapable that even for such a fixed, well defined body of mathematical propositions, *mathematical thinking is, and must remain, essentially creative.*

Gödel's [1965a, p. 5] incompleteness theorem is based on "Principia Mathematica and related systems". Referring to his result that there is no finite method deciding whether a sequence is generated by the operations of a normal system Post [1965a, pp. 407-408] writes:

... the analysis ... is fundamentally weak in its reliance on the logic of Principia Mathematica ... But for full generality a complete analysis would have to be made of all possible ways in which the human mind could set up finite processes for generating sequences.

Davis [1982b, p. 21] writes that Post "evidently felt that the very incompleteness of 'Principia Mathematica' ... undermined its suitability as a basis for such an analysis."

Turing's, Gödel's and Post's remarks suggest the possible existence of "mental" procedures that cannot be reduced to Turing machines. Section 2 introduces an effective procedure that computes a total function. We prove that a second input of the procedure cannot be set to the Gödel number of any Turing program that computes the output from any natural number in its first input. The procedure concerns the question whether every procedure used in an "intelligent" system can be modeled by a Turing machine. In particular, it concerns the question whether an "intelligent" system can model all its own functions by a Turing machine.

Section 3 uses the procedure in Section 2 to define creative procedures which compute functions that are not Turing-computable. We argue that creative procedures capture an aspect of Gödel's "mental procedures" which are not Turing-computable. In Section 4 we discuss Church's thesis.

2 Procedure

For his own form of Gödel's incompleteness theorem Post [1944] introduced creative sets whose definition implicitly refers to productive functions.¹ We

¹The term *productive* is due to Dekker [1955].

regard the existence of productive functions as a key to the phenomenon of incompleteness.

We deal with natural numbers, sets of natural numbers, and functions from natural numbers to natural numbers. We use the following terminology and notations: We write *domain* φ for the *domain* of a function φ and *range* φ for the *range* of φ . A function is called *partial* if its domain is a subset of the set all natural numbers. A function is called *total* if its domain is the set of all natural numbers. If φ is a partial function of natural numbers, we say that φ is *defined* at the natural number x if $x \in \text{domain } \varphi$.

Because Turing machines are represented as finite sets of instructions, that is, as finite sequences of a fixed finite number of symbols, it is possible to list the sets of instructions of all Turing machines by an algorithm, for example, in ascending length according to the number of symbols that a set of instructions contains. We follow Rogers [1987, p. 21] and keep such a listing fixed for the remainder of this article:

Definition 1. The *Turing program* P_i is the set of instructions of a Turing machine associated with the natural number i in a fixed listing of the sets of instructions of all Turing machines. i is called the *index* or *Gödel number* of P_i . φ_i is the partial function determined by P_i . i is also called the *index* or *Gödel number* of φ_i .

The listing gives an algorithm for generating P_i from any natural number i and another algorithm for generating a natural number i from the set of instructions P of any Turing machine such that P is P_i . The two algorithms can be encoded as ordinary computer programs.

A set of natural numbers is recursively enumerable if there is an algorithm for enumerating its members. A precise definition is [Rogers, 1987, p. 58]:

Definition 2. A set of natural numbers is *recursively enumerable* if it is empty or the range of a Turing-computable total function.

A set is called productive if there is a mechanical procedure (algorithm) which, given any recursively enumerable subset, produces a member of the set that is not contained in the given subset. A precise definition is:

Definition 3. A set S of natural numbers is *productive* if there is a Turing-computable partial function ψ such that, given any total function φ_j whose range is a subset of S , the value $\psi(j)$ is defined and contained in S but not in the range of φ_j :

$$(\forall j)[\varphi_j \text{ total} \ \& \ \text{range } \varphi_j \subseteq S \Rightarrow \\ [\psi(j) \text{ defined} \ \& \ \psi(j) \in S - \text{range } \varphi_j]]$$

The partial function ψ is called a *productive partial function* for S .

Definition 3 is equivalent to the definition of productive sets in Rogers [1987, p. 84, see p. 90] because of basic theorems such as Rogers [1987, p. 60, Theorem V, and p. 61, Corollary V(b)].

The following theorem states that the set $\{i | \varphi_i \text{ total}\}$ of the Gödel numbers i of all (Turing-computable) total functions φ_i is productive. According to Definition 3, this means that there is a Turing-computable partial function ψ , which is called productive, such that, given any (Turing-computable) total function φ_j with $\text{range } \varphi_j \subseteq \{i | \varphi_i \text{ total}\}$, the value $\psi(j)$ is defined and contained in $\{i | \varphi_i \text{ total}\}$ but not in the range of φ_j .

Theorem 1. *The set $\{i | \varphi_i \text{ total}\}$ of the Gödel numbers i of all total functions φ_i is productive.*

Proof. Let j be any natural number such that φ_j is a total function with $\text{range } \varphi_j \subseteq \{i | \varphi_i \text{ total}\}$, that is, the range of φ_j is a set of Gödel numbers of Turing-computable total functions. Thus, $\varphi_{\varphi_j(n)}$ is a Turing-computable total function for any natural number n . Using Cantor's diagonal method we define a new function δ_j by

$$\delta_j(n) = \varphi_{\varphi_j(n)}(n) + 1 \tag{1}$$

for all natural numbers n . Obviously, δ_j is a total function because $\varphi_{\varphi_j(n)}$ is a total function for any natural number n .

In order to prove the theorem, we construct a Turing-computable procedure computing a partial function ψ whose input is any natural number j satisfying the properties that φ_j is a total and $\text{range } \varphi_j \subseteq \{i | \varphi_i \text{ total}\}$. Such a natural number j is given at the beginning of the proof and is used in the definition of the function δ_j in (1). The output $\psi(j)$ of the function ψ for the input j is the Gödel number $\psi(j)$ of a Turing program $P_{\psi(j)}$ computing the function δ_j in (1). The Gödel number $\psi(j)$ is constructed as follows: According to Definition 1, the function φ_j in (1) is computed by the Turing program P_j and the function $\varphi_{\varphi_j(n)}$ in (1) is computed by the Turing program $P_{\varphi_j(n)}$. The expression $\varphi_{\varphi_j(n)}(n) + 1$ in (1) can be regarded as a pseudo-code for a Turing program $P_{\psi(j)}$ which computes the function δ_j in (1) and can be constructed from the Turing programs P_j and $P_{\varphi_j(n)}$, where

$\varphi_j(n)$ is the result of applying P_j to n . The construction of $P_{\psi(j)}$ from P_j and $P_{\varphi_j(n)}$ can be achieved by a Turing-computable procedure that is independent of the Gödel number j because the states of P_j and $P_{\varphi_j(n)}$ (called m -configurations, that is, machine configurations, in Turing [1936] and internal states in Rogers [1987, p. 13]) can be renamed such that P_j and $P_{\varphi_j(n)}$ only contain different states and can thus be used as components of $P_{\psi(j)}$. Because of Definition 1, the Gödel number $\psi(j)$ of $P_{\psi(j)}$ can be generated from $P_{\psi(j)}$ by a Turing-computable procedure. From the Turing-computable procedures producing $\psi(j)$ from $P_{\psi(j)}$ and $P_{\psi(j)}$ from P_j and $P_{\varphi_j(n)}$ we can construct a Turing-computable procedure that computes $\psi(j)$ from the Gödel number j of any total function φ_j with $\text{range } \varphi_j \subseteq \{i | \varphi_i \text{ total}\}$. Therefore, ψ is a Turing-computable partial function that computes the Gödel number $\psi(j)$ of a Turing program $P_{\psi(j)}$ computing the total function δ_j from the Gödel number j of any total function φ_j with $\text{range } \varphi_j \subseteq \{i | \varphi_i \text{ total}\}$ which was given at the beginning of the proof.

Because of (1) the function δ_j is different from $\varphi_{\varphi_j(n)}$ for all natural numbers n . Because the Turing program $P_{\psi(j)}$, which computes $\varphi_{\psi(j)}$ according to Definition 1, also computes the function δ_j in (1), $\delta_j(n) = \varphi_{\psi(j)}(n)$ for all natural numbers n . Therefore, the total function $\varphi_{\psi(j)}$ is different from $\varphi_{\varphi_j(n)}$ for all natural numbers n . In particular, $\psi(j)$ is different from $\varphi_j(n)$ for all natural numbers n because different functions $\varphi_{\psi(j)}$ and $\varphi_{\varphi_j(n)}$ cannot have the same Gödel number according to Definition 1. This implies that $\psi(j)$ is not contained in $\text{range } \varphi_j$, that is, the range of φ_j .

Therefore, ψ is a productive partial function for the set $\{i | \varphi_i \text{ total}\}$ of the Gödel numbers i of all total functions φ_i . In view of Definition 3 this completes the proof of the theorem. \square

Rogers [1987, p. 84, *Example 2*] suggests another proof that the set $\{i | \varphi_i \text{ total}\}$ is productive.

Definition 4. A recursively enumerable set S of natural numbers is *creative* if its complement is productive.

Referring to Gödel's [1965a] incompleteness theorem, Rogers [1987, pp. 97-98] states that [the Gödel numbers of] the provable well-formed formulas of Peano arithmetic form a creative set, [the Gödel numbers of] the unprovable well-formed formulas form a productive set, and [the Gödel numbers of] the

true well-formed formulas of elementary arithmetic form a productive set. Rogers [1987, p. 98] writes:

... no axiomatization of mathematics can exactly capture all true statements in elementary arithmetic; and from any axiomatization which yields only true statements in elementary arithmetic, a new true statement can be found not provable in that axiomatization.

... Post believed that such facts manifest an essentially creative quality of mathematics; hence the name *creative set*.

Soare [1978, p. 1151] writes:

Such r.e. [recursively enumerable] sets were called creative by Post ... because their existence ... implies the impossibility of mechanically listing all statements true in such a fragment [fragment of mathematics as elementary number theory].

The construction of undecidable formulas of formal systems in Gödel's [1965a] incompleteness theorem can be represented by an algorithm. Productive functions (see Definition 3), which are Turing-computable, correspond to a formal abstraction of this algorithm. Thus, the input of productive functions corresponds to formal systems. This suggests that formal systems, in particular, Turing programs, cannot refer to themselves, that is, they cannot capture their own *existence*. Otherwise, the application of productive functions, which correspond to the construction of the undecidable propositions, could be used to “overcome” incompleteness as described below.

We use any productive function ψ for the set of the Gödel numbers of all Turing-computable total functions in Theorem 1 to define a procedure Q which computes the output $\omega(x)$ of a total function ω for any natural number x in its input and prove that function ω is not Turing-computable. The procedure Q has a *second input* j which is in the domain of ψ and contains the Gödel number of a Turing program representing an *existing* formal system. The function ω is not Turing-computable because Gödel numbers $\psi(j)$, where j is an *existing* Gödel number in the domain of ψ , are contained in the output of the procedure Q , that is, the output of ω . Roughly speaking, there is no Gödel number j of a Turing program P_j generating all Gödel numbers of Turing-computable total functions that a human (or a “machine”) generates if the human (or the “machine”) applies ψ to the *existing* Gödel number

1. $l := \text{least } \{n \in \mathbf{N} \mid n \notin \text{domain } \alpha\};$
2. *if* $\text{is-set } (j)$
3. *then* $\alpha := \alpha \cup \{(l, \psi(j))\};$
4. *if* $\text{is-not-set } (x)$
5. *then* $\text{return } 1;$
6. *if* $x \in \text{domain } \alpha$
7. *then* $\text{return } \alpha(x);$
8. $\alpha := \alpha \cup \{(x, c)\};$
9. $\text{return } \alpha(x);$

Table 1: *Procedure Q with two inputs x and j and a global variable α for computing $\omega(x)$*

j and thus produces the Gödel number $\psi(j)$ of a Turing-computable total function that is not generated by P_j .

Definition 5. The procedure Q , whose pseudo-code is given in Table 1, has two input variables x and j . The variable x is not set or set to any natural number, that is, x has no value or the value of x is any natural number. The variable j is not set or set to the Gödel number of any Turing-computable total function whose range is a subset of the set $\{i \mid \varphi_i \text{ total}\}$ of the Gödel numbers i of all total functions φ_i .

The global variable α in the procedure Q in Table 1 is set to the empty set \emptyset before the first execution of the procedure Q . The variable α is a function which is represented as a set of input-output pairs (x, y) , that is, $(x, y) \in \alpha$ means $\alpha(x) = y$ in ordinary notation. The variable α is only changed by the procedure Q itself.

The first line of the procedure Q in Table 1 sets the variable l to the least natural number $n \in \mathbf{N}$ that is not contained in the domain of the function α which was set to the empty set \emptyset before the first execution of the procedure Q . The function ψ in the third line of the procedure Q is a productive partial function for the productive set $\{i \mid \varphi_i \text{ total}\}$ of the Gödel numbers i of all total functions φ_i . Such a productive partial function ψ exists according to Theorem 1 (see Definition 3). If the second input variable j is set (has a value), that is, the condition in the second line of the procedure Q is satisfied, the third line adds the input-output pair $(l, \psi(j))$ to the function α , which is represented as a set of input-output pairs. If the first input variable x is not set (has no value), that is, the condition in the fourth line of Q is satisfied,

the fifth line returns 1 as the output of the procedure Q . If the condition $x \in \text{domain } \alpha$ in the sixth line is satisfied, the procedure Q returns $\alpha(x)$ as its output in the seventh line. Otherwise, the eighth line adds the input-output pair (x, c) to the function α , where the constant c is the Gödel number of any fixed Turing-computable total function. Finally, the ninth line returns $\alpha(x)$, which is equal to c because of the eighth line, as the output of the procedure Q .

Theorem 2. All elementary operations of the procedure Q in Definition 5 and Table 1 are Turing-computable.

Proof. The least natural number $n \in \mathbf{N}$ that is not contained in the domain of the function α in the first line in Table 1 is Turing-computable because α is a finite set of input-output pairs at every point in time. The expression $\psi(j)$ in the second line in Table 1 is Turing-computable because j is the Gödel number of any Turing-computable total function whose range is a subset of the set $\{i \mid \varphi_i \text{ total}\}$ of the Gödel numbers i of all total functions φ_i according to Definition 5, j is in the domain of ψ according to Definition 3 and Theorem 1, and ψ is Turing-computable according to Definition 3 and Theorem 1. Obviously, the other elementary operations in the procedure Q are also Turing-computable. \square

Theorem 3. The procedure Q in Definition 5 and Table 1 computes a total function whose input is any natural number x in the first input of Q and whose output is the output of Q , where the second input variable j of Q is not set or set and may be changed at any time according to Definition 5.

Proof. The value $\alpha(x)$, where x is any natural number, in the seventh and the ninth line in Table 1 is uniquely determined by the set α of input-output pairs representing the function α : Let x , y_1 , and y_2 be any natural numbers with $(x, y_1) \in \alpha$ and $(x, y_2) \in \alpha$. This implies $y_1 = y_2$ because of the construction of α in the first and the third line and the extension of α in the eighth line is only used if $x \notin \text{domain } \alpha$. Thus, the value $\alpha(x)$ is uniquely determined. Therefore, the value $\omega(x)$ is uniquely determined because $\omega(x) = \alpha(x)$ for any natural number $x \in \text{domain } \alpha$.

The procedure Q in Table 1 computes the value $\omega(x)$ of the function ω for any natural number x because the input-output pair (x, c) is added to α in the eighth line if $x \notin \text{domain } \alpha$. Therefore, the domain of the function ω , which is computed by the procedure Q , is the set of all natural numbers, that is, ω is a total function. \square

Definition 6. We write ω for the total function computed by the procedure Q according to Theorems 2 and 3.

Theorem 4. The range of the total function ω in Definition 6 is a subset of the set $\{i|\varphi_i \text{ total}\}$ of all Turing-computable total functions.

Proof. According to Definition 5 the second input j of the procedure Q in Table 1 is not set or set to the Gödel number of any Turing-computable total function whose range is a subset of $\{i|\varphi_i \text{ total}\}$. Thus, the value $\psi(j)$ in the third step of Q is the Gödel number of a Turing-computable total function. In the third step of Q , the value $\psi(j)$ is used as an output of the function α . According to Definition 5, the value c in the eighth step of Q is the Gödel number of a fixed Turing-computable total function. In the eighth step of Q , the value c is used as an output of the function α . Thus, the range of α is a subset of $\{i|\varphi_i \text{ total}\}$. This implies that range of ω is a subset of $\{i|\varphi_i \text{ total}\}$ because any member in the range of ω is contained in the range of α according to the seventh and the ninth step of Q . \square

The following examples illustrate the computation of the function ω by the procedure Q .

Example 1. According to Definition 5 the global variable α in the procedure Q in Table 1 is set to the empty set \emptyset before the first execution of the procedure Q .

In order to compute, for example, the output $\omega(1)$ of the input 1 we apply the procedure Q to the value 1 of its first input variable x . According to the eighth line in Table 1, the input-output pair $(1, c)$ is added to the function α which is represented as a set of input-output pairs. According to the ninth line in Table 1, the procedure Q returns

$$\omega(1) = \alpha(1) = c \tag{2}$$

as the output $\omega(1)$ of the input 1.

In order to compute the output $\omega(5)$ of the input 5 we apply the procedure Q to the value 5 of its first input variable x . Thus, the input-output pair $(5, c)$ is added to the function α and the procedure Q returns

$$\omega(5) = \alpha(5) = c \tag{3}$$

as the output $\omega(5)$ of the input 5.

Let j_1 be the Gödel number of any Turing-computable total function φ_{j_1} whose range is a subset of $\{i | \varphi_i \text{ total}\}$. We apply the procedure Q to the value j_1 of its second input variable j . Because the value of j is set, the third line in Table 1 adds the input-output pair $(2, \psi(j_1))$ to the function α , where $2 = \text{least } \{n \in \mathbf{N} \mid n \notin \text{domain } \alpha\}$ according to the first line in Table 1.

In order to compute the output $\omega(2)$ of the input 2 we apply the procedure Q to the value 2 of its first input variable x . Because $2 \in \text{domain } \alpha$, the procedure Q returns

$$\omega(2) = \alpha(2) = \psi(j_1) \quad (4)$$

as the output $\omega(2)$ of the input 2 according to the seventh line in Table 1.

Example 2. We set the global variable α in the procedure Q in Table 1 to the empty set \emptyset before the first execution of Q .

Let j_1 be the Gödel number of any Turing-computable total function φ_{j_1} whose range is a subset of $\{i | \varphi_i \text{ total}\}$. We apply the procedure Q to the value j_1 of its second input variable j . Thus, the third line in Table 1 adds the input-output pair $(1, \psi(j_1))$ to the function α , where 1 is the least natural number that is not contained in the domain of α according to the first line in Table 1. Then, the computation of the output $\omega(1)$ yields

$$\omega(1) = \alpha(1) = \psi(j_1). \quad (5)$$

We construct a sequence of Gödel numbers j_2, j_3, \dots of Turing-computable total functions $\varphi_{j_2}, \varphi_{j_3}, \dots$ as follows: Let $k > 1$ be any natural number. We define a function φ_{j_k} by $\varphi_{j_k}(1) = \psi(j_{k-1})$ and

$$\varphi_{j_k}(x) = \varphi_{j_{k-1}}(x - 1) \quad (6)$$

for all natural numbers $x > 1$. Thus, the range of φ_{j_k} contains $\psi(j_{k-1})$ and $\varphi_{j_{k-1}}(x)$ for any natural number x . Obviously, φ_{j_k} is a Turing-computable total function whose range is a subset of $\{i | \varphi_i \text{ total}\}$.

The application the procedure Q to the value j_k of its second input variable j adds the input-output pair $(k, \psi(j_k))$ to the function α . Therefore, the computation of the output $\omega(k)$ yields

$$\omega(k) = \alpha(k) = \psi(j_k) \quad (7)$$

for any natural number k .

The procedure Q in Table 1 contains two input variables x and j . If Q is applied to any natural number x , Q produces an output $\omega(x)$, that is, Q computes a total function. If ω were Turing-computable, there would be a Turing program computing ω . The proof of the following theorem shows that the input variable j of Q cannot be set to the Gödel number of this Turing program, that is, P_j , because the productive function ψ in Q is applied to j and $\psi(j)$ is used in the output of ω such that Theorem 1 precludes that P_j computes ω . The condition in the following theorem entails that $\psi(j)$ is in the output of the function ω according to the third line of the procedure Q . Therefore, the Turing program P_j does not compute ω because $\psi(j)$ is not contained in the output of P_j according to Theorem 1. An explanation is that the Gödel number $\psi(j)$ can be used to construct a more powerful Turing program that produces $\psi(j)$ and all Gödel numbers in the output of P_j . Thus, Q can be used to construct more and more powerful Turing programs.

Theorem 5. There is no Gödel number j such that the Turing program P_j computes the total function ω in Definition 6 if the procedure Q , which computes the function ω according to Theorems 2 and 3, is applied to the Gödel number j of the Turing program P_j in its second input.

Proof. We assume that there is any natural number j such that the Turing program P_j computes the total function ω in order to derive a contradiction. Our assumption implies that

$$\omega(x) = \varphi_j(x) \tag{8}$$

for all natural numbers x because P_j also computes φ_j according to Definition 1. According to the condition in the theorem to be proved we apply the procedure Q to the Gödel number j of the Turing program P_j in its second input. Because the range of the function ω is a subset of $\{i | \varphi_i \text{ total}\}$ according to Theorem 4, j is the Gödel number of a Turing-computable total function whose range is a subset of $\{i | \varphi_i \text{ total}\}$, that is, j is an admissible input of the procedure Q in Definition 5. Because the second input variable j of the procedure Q is set, that is, the condition in the second line of the procedure Q in Table 1 is satisfied, the third line adds the input-output pair $(l, \psi(j))$ to the function α , that is,

$$\alpha(l) = \psi(j), \tag{9}$$

where $l = \text{least } \{n \in \mathbf{N} \mid n \notin \text{domain } \alpha\}$ according to the first line in Table 1. Because of (9), $l \in \text{domain } \alpha$ and

$$\omega(l) = \alpha(l) = \psi(j) \quad (10)$$

because of the seventh line in Table 1. Because of (8),

$$\omega(l) = \varphi_j(l). \quad (11)$$

This implies that $\psi(j)$ is contained in the range of the function φ_j . According to Theorem 1 the value $\psi(j)$ is not contained in the range of the function φ_j . Thus, we have derived a contradiction from our original assumption that any Turing program P_j computes the total function ω . Therefore, there is no Turing program P_j , where j is any natural number, that computes the total function ω . \square

The condition "if the procedure $Q \dots$ " in Theorem 5 can be implemented by the following if-then rule:

Rule 1. If j is the Gödel number of a Turing program P_j that computes a total function whose range is a subset of $\{i \mid \varphi_i \text{ total}\}$, then apply the procedure Q in Definition 5 to x and j , where the value of the input variable x is not set.

If Rule 1 is applied, the condition "if the procedure $Q \dots$ in its second input" in Theorem 5 is satisfied. This implies that there is no Turing program P_j , where j is any natural number, computing the total function ω in Definition 6.

The condition in Rule 1 may be modified. For example, the condition may require a formal proof that P_j "computes a total function whose range is a subset of $\{i \mid \varphi_i \text{ total}\}$."

3 Discussion

The proofs of Theorems 2, 3, 4, and 5 in Section 2 constitute precise evidence that all elementary operations of the procedure Q in Table 1 are Turing-computable and the total function ω , which is computed by the procedure Q , is not Turing-computable. The theorems are independent of the special productive function ψ of the procedure Q in Table 1, that is, Theorems 2, 3, 4, and 5 remain valid if the procedure Q in Table 1 uses any productive function. This suggests the following definition:

Definition 7. A procedure, whose elementary operations are computable by Turing programs, is called *creative* if it computes functions that are not computable by Turing programs.

According to Theorems 2 and 3 the procedure Q computes the total function ω in Definition 6 which is not Turing-computable according to Theorem 5. This means that the procedure Q is creative in the sense of Definition 7.

Rule 1 in Section 2 provides Gödel numbers j of Turing programs P_j that compute total functions whose range is a subset of $\{i|\varphi_i \text{ total}\}$, that is, existing information, in the input of the procedure Q but these Gödel numbers j cannot completely be represented in Q because the set $\{i|\varphi_i \text{ total}\}$ is productive. The procedure Q is creative according to Definition 7 because Q computes the function ω which is not Turing-computable. Thus, Rule 1 is an implementation of the following general principle for creative procedures:

Existence Principle. Existing information, for example, Gödel numbers of Turing programs, is provided in the input of procedures which are creative according to Definition 7 if the information cannot completely be represented in the procedure.

The existence principle, in particular, Rule 1, can be implemented physically, that is, in a “machine”. Let

$$L(j), \tag{12}$$

where j is any natural number, be formal propositions that stand for the condition “ j is the Gödel number of a Turing program P_j that computes a total function whose range is a subset of $\{i|\varphi_i \text{ total}\}$ ” in Rule 1, that is, L in (12) refers to a fixed finite string. The formal proposition $L(j)$ has a physical representation. Rule 1 is applied if the proposition $L(j)$ in (12), in particular, the Gödel number j , is generated physically. The existence principle simply implies that existing information, which is available physically, is provided in the input of creative procedures such as the procedure Q in Table 1. In view of the results described previously, this means that the function ω can neither be modeled by any Turing machine nor be dealt within a formal system although the computation of ω can be implemented physically, that is, in a “machine”.

The function α in the procedure Q represents the function ω in the sense that the input-output pairs in α are a finite subset of the input-output pairs

of ω (see Definition 5 and Table 1 in Section 2). But there is no Gödel number for the function ω , which is computed by Q , because input-output pairs $(l, \psi(j))$ and (x, c) may be added to the function α whenever Q is executed.

Referring to his “Theorem 2.4, with its corollaries” Davis [1982a, pp. 121-122] writes:

... these results really constitute an abstract form of Gödel’s famous incompleteness theorem ... they imply that *an adequate development of the theory of natural numbers, within a logic L , to the point where membership in some given set Q of integers can be adequately dealt with within the logic ... is possible only if Q happens to be recursively enumerable*. Hence, non-recursively enumerable sets can, at best, be dealt with in an incomplete manner.

This implies that the function ω , which is computed by the procedure Q , cannot be dealt with within a logic, that is, a formal system because the range of ω is not recursively enumerable according to Theorem 5.

Let R be any Turing program that generates any sequence of Gödel numbers

$$j_1, j_2, \dots \tag{13}$$

of Turing programs P_{j_1}, P_{j_2}, \dots computing total functions whose range is a subset of $\{i | \varphi_i \text{ total}\}$, that is, R successively generates the Gödel numbers j_1, j_2, \dots in (13).² Rule 1 applies the procedure Q in Table 1 to any Gödel number in (13) as soon as such a Gödel number is generated. Rule 1 also processes Gödel numbers that are *not* generated by the Turing program R , that is, if the Gödel number, say j_k , of *any* Turing program P_{j_k} computing a total function is generated whose range is a subset of the set $\{i | \varphi_i \text{ total}\}$, then Rule 1 applies the procedure Q in Table 1 to j_k . Thus, according to Theorem 5 in Section 2 there is no Gödel number j_k such that P_{j_k} computes the total function ω in Definition 6.

If the Gödel numbers j in the input of Rule 1 are restricted to the Gödel numbers j_1, j_2, \dots in (13), which are generated by the Turing program R , the range of the total function ω in Theorem 5 is recursively enumerable. But such a restriction restricts the input of Rule 1 to a recursively enumerable

²The sequence of Gödel numbers j_1, j_2, \dots in Example 2 in Section 2 can be generated by such a Turing program R .

subset of the set $\{i|\varphi_i \text{ total}\}$ and thus restricts the generality of Rule 1 because the set $\{i|\varphi_i \text{ total}\}$ is productive. This means that the inputs of Rule 1 cannot be restricted to a recursively enumerable subset of $\{i|\varphi_i \text{ total}\}$ because a productive function ψ for $\{i|\varphi_i \text{ total}\}$ could be applied to a Gödel number j of a Turing program generating this subset and thus produce the Gödel number $\psi(j)$ of a Turing-computable function that is contained in $\{i|\varphi_i \text{ total}\}$ but not in this subset. Thus, the range of the total function ω in Theorem 5 is not recursively enumerable because the inputs of Rule 1 cannot be restricted to a recursively enumerable subset of the productive set $\{i|\varphi_i \text{ total}\}$.

Rogers [1987, pp. 10–11] discusses “the problem of getting a satisfactory [formal] characterization of algorithm and algorithmic function” because the application of diagonalization to a list of total algorithmic functions yields a total algorithmic function that is not contained in the list (see the proof of Theorem 1 in Section 2). Rogers [1987, pp. 11–12] writes:

We can avoid the diagonalization difficulty by allowing sets of instructions for nontotal partial functions as well as for total functions. ... The approach taken by way of partial functions is, in essence, the approach taken by Kleene ..., Church ..., Turing [1936] and others in the 1930’s.

Referring to “the concept of general recursiveness (or Turing’s computability)” Gödel [1965c, p. 84] writes:

... with this concept one has for the first time succeeded in giving an absolute definition of an interesting epistemological notion, i.e., one not depending on the formalism chosen. In all other cases treated previously, such as demonstrability ... it is clear that the one obtained is not the one looked for. ... By a kind of miracle ... the diagonal procedure does not lead outside the defined notion.

Thus, there is only a formal characterization of a single “interesting epistemological notion”, that is, the Turing-computable partial functions. In contrast, creative procedures provide a framework for an investigation of other “notions” such as Turing-computable total functions. This framework is not subject to a “diagonalization difficulty” but uses diagonalization to investigate such “notions” which cannot be captured by formal systems.

Discussing the formalization of a theory which results in a formal system, Kleene [1952, p. 64] writes:

Metamathematics must study the formal system as a system of symbols, etc. which are considered wholly objectively. This means simply that those symbols, etc. are themselves the ultimate objects, and are not being used to refer to something other than themselves. The metamathematician looks at them, not through and beyond them; thus they are objects without interpretation or meaning.

Referring to his undecidable formula $A_p(\mathbf{p})$ in Gödel's incompleteness theorem Kleene [1952, p. 426] writes:

... if we suppose the number-theoretic formal system to be consistent, we can recognize that $A_p(\mathbf{p})$ is true by taking into view the structure of that system as a whole, though we cannot recognize the truth of $A_p(\mathbf{p})$ by use only of the principles of inference formalized in that system, i.e. $\text{not } \vdash A_p(\mathbf{p})$.³

Thus, Gödel's theorem requires a reference to the (incomplete) formal “system as a whole” which cannot be achieved within the formal system itself because, in our interpretation, the formal system cannot take “into view the structure of that system as a whole”, that is, “wholly objectively”. In particular, a formal system represented by a Turing program cannot take “into view the structure of that” Turing program “as a whole”, that is, “wholly objectively”, in the sense that the Turing program, which produces a (recursively enumerable) subset of a productive set, cannot refer to itself and thus capture the result of applying a productive partial function for the productive set to its own Gödel number.

The assumption that all reasoning processes can be modeled by a Turing program (see Section 1) immediately yields a contradiction if the following if-then rule, which is also an implementation of the existence principle, is used:

Rule 2. If j is the Gödel number of a Turing program P_j that computes a total function whose range is a subset of $\{i \mid \varphi_i \text{ total}\}$, then $\psi(j)$, where ψ

³The expression “ $\text{not } \vdash A_p(\mathbf{p})$ ” in Kleene [1952] means that the undecidable formula $A_p(\mathbf{p})$ in Gödel's theorem is not provable in the formal system.

is a productive partial function for $\{i|\varphi_i \text{ total}\}$, is the Gödel number of a Turing-computable total function, that is, $\varphi_{\psi(j)}$ is a total function.

Rule 2 is an immediate implication of Theorem 1 in Section 2. A pseudo-code for Rule 2 is

$$\text{if } L(j) \text{ then } T(\psi(j)), \quad (14)$$

where $L(j)$ is a formal proposition which stands for the condition “ j is the Gödel number of a Turing program P_j ... a subset of $\{i|\varphi_i \text{ total}\}$ ” in Rule 2 and $T(j)$ is a formal proposition which stands for the consequent “ $\psi(j)$... is the Gödel number of a Turing-computable total function” in Rule 2, that is, L and T in (14) refer to fixed finite strings. If we assume that all reasoning processes can be modeled by a Turing program, say P_k , then P_k computes a total function whose range is the subset of $\{i|\varphi_i \text{ total}\}$ containing all members of $\{i|\varphi_i \text{ total}\}$ that are generated by P_k . Let j be the Gödel number j of a Turing program P_j that computes this total function. The use of Rule 2, which is an implementation of the existence principle, implies that a formal representation (14) of Rule 2 and a formal representation $L(t)$ of its condition are contained in the Turing program P_k which is assumed to model all reasoning processes. The application of (14) to $L(t)$ by the Turing program P_k yields the consequent $T(\psi(j))$ in (14). Thus, we have a contradiction because, according to Theorem 1 in Section 2, $\psi(j)$ is not contained in the range of φ_j which contains all members of $\{i|\varphi_i \text{ total}\}$ that are generated by P_k . These considerations also apply to the “reasoning processes” of a “machine” or “robot” because, as described above, the existence principle, in particular, Rules 1 and 2, can be implemented physically.

As discussed above, a restriction of the input of Rule 1 to a recursively enumerable subset of $\{i|\varphi_i \text{ total}\}$ restricts the generality of Rule 1. The use of Rule 2, which is an implementation of the existence principle, also implies that such a restriction restricts the generality of Rule 1 because the application of Rule 2 to the Gödel number j of a Turing-computable total function whose range is the subset of $\{i|\varphi_i \text{ total}\}$ yields the Gödel number $\psi(j)$ of a Turing-computable total function that is not contained in this subset but satisfies the condition in Rule 1. A proof that j is the Gödel number of a Turing-computable total function whose range is the subset of $\{i|\varphi_i \text{ total}\}$ can be extended to a proof that $\psi(j)$ is the Gödel number of a Turing-computable total function and not contained in this subset because ψ is a productive partial function.

No formal system can capture its own existence in the sense that every Turing program representing a formal system and generating (recursively enumerable) subsets of productive sets cannot generate the result of applying productive partial functions for the productive sets to the Gödel numbers of Turing programs producing these subsets (see Theorem 1 in Section 2 and the quotation from Davis [1982a, pp. 121-122] above). This implies that a Turing program generating a (recursively enumerable) subset of a productive set cannot contain a reference to its own Gödel number because a productive function for the productive set could be applied to such a reference such that the original Turing program could contain the output of the productive function. The incompleteness of formal systems is “overcome” by the existence principle which can be implemented in rules such as Rule 1. Roughly speaking, this principle implies that existing information is provided in the input of creative procedures and thus “overcomes” the incompleteness of formal systems and the limits of Turing’s computability.⁴ Thus, the second input variable j in the procedure Q in Table 1 is required because formal systems including Turing programs cannot refer to existing information such as their own Gödel numbers.

4 Related Work

Church’s [1965, pp. 90, 100-102] thesis⁵ states that every effectively calculable function is general recursive, that is, computable by a Turing machine (see Kleene [1952, pp. 300–301, 317–323]). Since “effective calculability” is an intuitive concept, the thesis cannot be proved (see Kleene [1952, p. 317]).⁶

Referring to Gödel’s [1965a] incompleteness theorem and Church’s [1965, pp. 90, 100-102] identification of effective calculability with recursiveness, Post [1965b, p. 291, footnote 8] writes:

“Actually the work already done by Church and others carries this identification considerably beyond the working hypothesis stage.

⁴Post [1965a, p. 423] writes: “What we must now do is to isolate the creative germ in the thinking process.”

⁵The term *Church’s thesis* is due to Kleene [1965, p. 274] (see Kleene [1952, pp. 300, 317]).

⁶In his article “Why Gödel Didn’t Have Church’s Thesis” Davis [1982b, p. 22, footnote 26] writes: “We are not concerned here with attempts to distinguish ‘mechanical procedures’ (to which Church’s thesis is held to apply) from a possible broader class of ‘effective procedures’ ...”

But to mask this identification under a definition hides the fact that a fundamental discovery in the limitations of the mathematizing power of Homo Sapiens has been made and blinds us to the need of continual verification.”

Thus, Post calls for a “continual verification” of Church’s thesis because of the incompleteness of formal systems (see Section 1).

Referring to Principia Mathematica (see Gödel [1965a]) and his normal systems, Post [1965a, p. 408] writes (see Section 1):

... for full generality a complete analysis would have to be made of all possible ways in which the human mind could set up finite processes for generating sequences.

In our view, the existence principle can be used by the “human mind” because existing information, for example, a Turing machine representing a formal system, must be a constituent of the “human mind”, that is, a constituent of reasoning. Thus, the “human mind” can apply an implementation of the existence principle, for example, Rule 1, and the procedure Q in Table 1 in Section 2 to “set up finite processes for generating sequences” which cannot be computed by any Turing program according to Theorem 5. This means that a formal system cannot deal with a fundamental aspect of reasoning because it cannot refer to its own existence.

The function ω , which is computed by the procedure Q in Table 1 in Section 2, can be regarded as effectively calculable because the elementary operations of Q are Turing-computable according to Theorem 2 and Rule 1, which uses the procedure Q , can be implemented physically.

The condition “if the procedure Q ...” in Theorem 5 is satisfied if Rule 1 in Section 3 is applied. Rule 1 is an implementation of the existence principle, which states that existing information, for example, Gödel numbers of Turing-computable total functions whose range is a subset of all Turing-computable total functions, that is, recursively enumerable subsets of a productive set, is provided in the input of creative procedures which contain a productive function for the productive set. Thus, Church’s thesis is not valid if the existence principle is applied to recursively enumerable subsets of productive sets and suitable procedures.

Church [1965, pp. 90, 102] presents his thesis as a “definition of effective calculability” and proposes a second definition of effective calculability:

... (2) by defining a function F (of positive integers) to be effectively calculable if, for every positive integer m , there exists a positive integer n such that $F(m) = n$ is a provable theorem.

If we require for every input j of the procedure Q in Table 1 a formal proof that the Turing program P_j computes a total function (see the modification of Rule 1 in Section 2), then, for every natural number (positive integer) x in the input of Q , which computes the function $\omega(x)$, there exists a natural number y such that $\omega(x) = y$ is a provable theorem in some formal system, say S_x . Such a formal system also exists for any finite set of natural numbers x in the input of Q . But, because of Theorem 5 in Section 2, there exists no formal system S such that $\omega(x) = y$, where y is a natural number, is a provable theorem in S for all natural numbers x .⁷ Roughly speaking, Theorem 5 implies that the formal systems S_x cannot be unified into a single formal system S .

In a letter of June 8, 1937, to Pepis Church wrote (see Sieg [1997, pp. 175–176]):

... if a numerical function f is effectively calculable then for every positive integer a there must exist a positive integer b such that a valid proof can be given of the proposition $f(a) = b$...

Therefore to discover a function which was effectively calculable but not general recursive would imply discovery of an utterly new principle of logic, not only never before formulated, but never before actually used in a mathematical proof - since all extant mathematics is formalizable within the system of Principia [Mathematica], or at least within one of its known extensions.

As far as we know the existence principle was “never before actually used in a mathematical proof”.

The function α in the procedure Q in Definition 5 and Table 1, which is represented as a set of input-output pairs, is a subset of the input-output pairs of the function ω . Q returns $\alpha(x)$ as the output of $\omega(x)$ for any natural number x in the first input of Q , that is, the input of ω . Nevertheless, the set α of input-output pairs, which is the empty set \emptyset before the first

⁷Here, we implicitly assume that the formal system S is represented by a Turing machine and a Gödel number for the recursively enumerable subset of $\{i | \varphi_i \text{ total}\}$ produced by S is generated such that Theorem 5 and Rule 1 are applicable.

execution of Q , is finite at every point in time. Obviously, each set α of input-output pairs can be generated by a Turing machine. Thus, the number of states of Turing programs generating α is finite “at each stage” of its development but according to Theorem 5 there is no Gödel number of a Turing program computing ω , that is, α “at each stage” of its development (see Gödel [1990, p. 306] and Section 1). This means that the number of states of the Turing machines generating α will not be “confused” by an “infinity of states of mind” because this number of states is finite “at each stage” of the development of α , that is, the development of ω (see Turing [1936, pp. 249–250] and Section 1).⁸

Referring to Church’s thesis and Gödel’s “mental procedures” Kleene [1987, pp. 493, 494] writes:

For, in the idea of “effective calculability” or of an “algorithm” as I understand it, it is essential that all of the infinitely many calculations ... are performable ... by following a set of instructions fixed in advance of all the calculations. If the Turing machine representation is used, this includes there being only a finite number of “internal machine configurations”, corresponding to a finite number of a human computer’s mental states. ... As Turing ... says ..., “If we admitted an infinity of states of mind, some of them will be ‘arbitrarily close’ and will be confused.” Hardly appropriate for keeping things straight digitally!

... an effective (finitely describable) procedure from the beginning, coming under the Church-Turing thesis.

In our view, the descriptions of the procedure Q in Section 2 and Rule 1 in Section 3 are “fixed” at the beginning and “finite” at every point in time but

⁸We suppose Turing’s [1936, pp. 249–250] infinity is an infinity according to a mathematical definition. For example, a definition states that a set is *infinite* if it is not finite. Another definition states that a set A is *Dedekind-infinite* if some proper subset B of A is equinumerous to A , that is, there is a bijection (one-to-correspondence) between A and B . In Zermelo-Fraenkel set theory this definition is equivalent to the condition that a set A is infinite if there is a one-to-one correspondence between all natural numbers and a subset of A . The number of states of the Turing machines generating α is not finite and seems to be “unbounded” because there is no Turing machine generating α “at each stage” of its development if Rule 1 is applied. Hilbert [1983, pp. 183–186] writes: “... the *infinite*, as that concept is used in mathematics, has never been completely clarified ... the infinity in the sense of an infinite totality, where we still find it used in deductive methods, is an illusion. ... no other *concept* needs *clarification* more it does.”

the function α in the procedure Q develops in the course of time because it depends on the input variables x and j of Q , that is, the functions α and ω cannot be described "in advance" because the set of the Gödel numbers of the Turing-computable total functions is productive. As described above, the number of states of Turing machines generating the function α will not be "confused" by an "infinity of states of mind" because this number of states is finite "at each stage" of the development of α , that is, the development of ω (see Turing [1936, pp. 249–250] and Section 1).

The proof of Theorem 1 in Section 2 states that the Gödel number $\psi(j)$ of the total function $\varphi_{\psi(j)}$ is not contained in the range of φ_j , where j is any natural number such that φ_j is a total function with $\text{range } \varphi_j \subseteq \{i | \varphi_i \text{ total}\}$. In particular, the proof states that the total function $\varphi_{\psi(j)}$ is different from $\varphi_{\varphi_j(n)}$ for all natural numbers n . This means that the productive function ψ produces not only a new Gödel number $\psi(j)$, which is different from $\varphi_j(n)$ for all natural numbers n , but also a new total function $\varphi_{\psi(j)}$, which is different from all total functions $\varphi_{\varphi_j(n)}$ for all natural numbers n .

Gödel [1990, p. 306] writes (see Section 1):

... *mind ... is not static, but constantly developing, ...*

If creative procedures and the existence principle are used, "*mind*" is "*constantly developing*" in the sense that it produces new structures from existing structures. For example, the use of Rule 1, which is an implementation of the existence principle, produces a new Gödel number $\psi(j)$ in the creative procedure Q whenever any natural number j such that φ_j is a total function with $\text{range } \varphi_j \subseteq \{i | \varphi_i \text{ total}\}$ is generated. As described above, $\varphi_{\psi(j)}$ is a new total function which is different from all total functions $\varphi_{\varphi_j(n)}$ for all natural numbers n .⁹

⁹All existing structures in a creative procedure are called "reflection base". The repeated application of the existence principle, that is, the repeated application of a creative procedure to information in its reflection base, can be regarded as a feedback process which produces more and more powerful structures. A first step towards the implementation of a creative procedure is described in Ammon [1988], Ammon [1992], and Ammon [1993]. These experiments suggest that creative procedures are a self-developing process which can start from any universal programming language. Their structure can be regarded as a web of concepts and methods which are called "analytical spaces" and cannot be characterized formally. This is plausible because formal systems are restricted to Turing-computable partial functions, that is, recursively enumerable sets (see the quotations from Davis [1982a, pp. 121–122], Rogers [1987, pp. 11–12], and Gödel [1965c, p. 84] in Section 3).

Lucas [1961] argues that mind cannot be modeled by a Turing machine because he *knows* that the undecidable proposition in Gödel’s theorem is true (see Shapiro [1998, pp. 273-274]). Putnam points out that Lucas cannot prove the prerequisite of consistency in Gödel’s theorem (see Shapiro [1998, pp. 282-284]). The procedure Q and Rule 1 in Section 2 can be executed by a human and by a “machine” to compute the function ω . According to Theorem 5 there is no Turing machine computing ω . Thus, an implementation of the existence principle, which merely uses the existence of a formal system, that is, a Turing machine itself, “overcomes” its incompleteness, that is, the limits of the Turing machine.¹⁰ Referring to Penrose [1990], Davis [1993, p. 611] writes:

However, it [Gödel’s theorem] is a quite ordinary sentence of elementary number theory and can be proved with no difficulty whatever in any formal system adequate for elementary number theory, such as for example Peano arithmetic. Note that this powerful form of Gödel’s theorem applies uniformly to any formalism whatever.

Although Gödel’s theorem applies to any (given) “formalism”¹¹ and can be proved in a “formal system”, no proof of Gödel’s theorem in any “formal system” can apply to this “formal system” itself because, as described above, no “formal system” can refer to itself, that is, to its own existence. Rather, a proof of Gödel’s theorem for a “formalism”, that is, “formal system”, say S_1 , requires another extended “formal system”, say S_2 , which refers to S_1 , a proof of Gödel’s theorem for S_2 , requires another extended “formal system”, say S_3 , which refers to S_2 , and so on.¹² Thus, there is no proof in any formal

¹⁰Gödel [1965b, pp. 71–72] writes: “... due to A.M. Turing’s work, a precise and unquestionably adequate definition of the general concept of formal system can now be given, ... Turing’s work gives an analysis of the concept of ‘mechanical procedure’ (alias ‘algorithm’ ...). This concept is shown to be equivalent with that of a ‘Turing machine’. A formal system can simply be defined to be any mechanical procedure for producing formulas, called provable formulas.

¹¹Originally, Gödel [1965a] proved his theorem for the “formalism” of “Principia Mathematica and related systems”.

¹²In particular, a reference to “any formalism whatever” cannot be formalized. For example, a Turing program generating a (recursively enumerable) subset of a productive set, is a “formalism”, that is, a formal system. A formal reference to all these Turing programs does not exist because productive sets are not recursively enumerable. Mendelson [1964, p. 253] writes that there are 2^{\aleph_0} productive sets, that is, the cardinality of the productive

system showing that Gödel’s theorem applies “to any formalism whatever”. Therefore, the “insight” that Gödel’s theorem applies “to any formalism whatever” requires a new principle such as the existence principle which cannot be formalized although it can be implemented physically.

5 Conclusion

We described a procedure that computes a total function whose range can contain members of a productive set. The elementary operations of the procedure are Turing-computable. We proved that there is no Turing program computing this total function if the existence principle is used which implies that existing recursively enumerable subsets of the productive set are provided in the second input of the procedure. The existence principle can be implemented in rules which can be executed by humans and “machines”. Roughly speaking, a formal system cannot contain a reference to itself and an extension of the recursively enumerable sets that it deals with by productive functions. In view of the existence principle, this means that a formal system cannot deal with a fundamental aspect of reasoning, that is, it cannot capture its own existence. Church’s thesis is not valid if the existence principle is applied to recursively enumerable subsets of productive sets and suitable procedures. We defined creative procedures which compute functions that are not computable by Turing machines and argued that creative procedures model an aspect of reasoning that cannot be modeled by Turing machines.

Acknowledgments. The author wishes to thank colleagues, in particular, Sebastian Stier and Andreas Keller, for helpful comments on earlier versions of this paper.

References

- [Ammon, 1988] K. Ammon. The automatic acquisition of proof methods. In *National Conference on Artificial Intelligence, St. Paul*, San Mateo, Calif., 1988. Morgan Kaufmann.

sets corresponds to the cardinality of the continuum. This also implies that a reference to “any formalism whatever” cannot be formalized.

- [Ammon, 1992] K. Ammon. Automatic proofs in mathematical logic and analysis. In *11th International Conference on Automated Deduction, Saratoga Springs*, pages 4–19, Berlin, 1992. Springer.
- [Ammon, 1993] K. Ammon. An automatic proof of Gödel’s incompleteness theorem. *Artificial Intelligence*, 61(2):291–306, 1993.
- [Church, 1965] A. Church. An unsolvable problem of elementary number theory. In M. Davis, editor, *The Undecidable*, pages 89–107. Raven Press, New York, 1965. Reprinted from *The American Journal of Mathematics*, vol. 58, pp. 345–363 (1936).
- [Davis, 1982a] M. Davis. *Computability and Unsolvability*. Dover, New York, 1982.
- [Davis, 1982b] M. Davis. Why Gödel didn’t have Church’s thesis. *Information and Control*, 54:3–24, 1982.
- [Davis, 1993] M. Davis. How subtle is Gödel’s theorem? More on Roger Penrose. *Behavioral and Brain Sciences*, 16:611–612, 1993.
- [Dekker, 1955] J. Dekker. Productive sets. *Transactions of the American Mathematical Society*, 78(1):129–149, 1955.
- [Gödel, 1965a] K. Gödel. On formally undecidable propositions of the Principia Mathematica and related systems I. In M. Davis, editor, *The Undecidable*, pages 4–38. Raven Press, New York, 1965. The original German title is: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, vol. 38 (1931), pp. 173–198.
- [Gödel, 1965b] K. Gödel. On undecidable propositions of formal mathematical systems - POSTSCRIPTUM. In M. Davis, editor, *The Undecidable*, pages 39–74. Raven Press, New York, 1965.
- [Gödel, 1965c] K. Gödel. Remarks before the princeton bicentennial conference on problems in mathematics. In M. Davis, editor, *The Undecidable*, pages 84–88. Raven Press, New York, 1965.
- [Gödel, 1990] K. Gödel. Gödel 1972a: Some remarks on the undecidability results. In S. Feferman et al., editors, *Collected Works: Publications 1938–1974*, volume 2. Oxford University Press, New York, 1990.

- [Hilbert, 1983] D. Hilbert. On the infinite. In P. Benacerraf and H. Putnam, editors, *Philosophy of Mathematics: Selected Readings*, pages 183–201. Cambridge University Press, Cambridge, 1983. Translated by Erna Putnam and Gerald J. Massey from *Mathematische Annalen* (Berlin) vol. 95 (1926), pp. 161–190.
- [Kleene, 1952] S. C. Kleene. *Introduction to Metamathematics*. North-Holland, Amsterdam, 1952.
- [Kleene, 1965] S. C. Kleene. Recursive predicates and quantifiers. In M. Davis, editor, *The Undecidable*, pages 255–287. Raven Press, New York, 1965. Reprinted from *Transactions of the American Mathematical Society*, Volume 53 (1943), No. 1, pages 41–73.
- [Kleene, 1987] S. C. Kleene. Reflections on Church’s thesis. *Notre Dame Journal of Formal Logic*, 28(4):490–498, 1987. Available at <http://projecteuclid.org/euclid.ndjfl/1093637645> (viewed Jan. 7, 2011).
- [Lucas, 1961] J. R. Lucas. Minds, machines, and Gödel. *Philosophy*, 36:112–137, 1961.
- [Mendelson, 1964] E. Mendelson. *Introduction to Mathematical Logic*. Van Nostrand Reinhold Company, New York, 1964.
- [Penrose, 1990] R. Penrose. Author’s response: the nonalgorithmic mind. *Behavioral and Brain Science*, 13:692–705, 1990.
- [Post, 1944] E. Post. Recursively enumerable sets of positive integers and their decision problems. *Bulletin of the American Mathematical Society*, 50(5):284–316, 1944. Available at <http://www.ams.org/journals/bull/1944-50-05/S0002-9904-1944-08111-1> (viewed Dec. 30, 2011).
- [Post, 1965a] E. Post. Absolutely unsolvable problems and relatively undecidable propositions - account of an anticipation. In M. Davis, editor, *The Undecidable*, pages 338–433. Raven Press, New York, 1965.
- [Post, 1965b] E. Post. Finite combinatory processes. Formulation I. In M. Davis, editor, *The Undecidable*, pages 289–291. Raven Press, New York,

1965. Reprinted from *The Journal of Symbolic Logic*, vol. 1 (1936), pp. 103-105.
- [Rogers, 1987] H. Rogers. *Theory of Recursive Functions and Effective Computability*. The MIT Press, Cambridge, 1987.
- [Shapiro, 1998] S. Shapiro. Incompleteness, mechanism, and optimism. *The Bulletin of Symbolic Logic*, 4(3):273–302, September 1998. Available at <http://www.math.ucla.edu/~asl/bsl/0403-toc.htm> (viewed Jan. 1, 2012).
- [Sieg, 1997] W. Sieg. Step by recursive step: Church’s analysis of effective calculability. *The Bulletin of Symbolic Logic*, 3(2):154–180, 1997. Available at <http://www.math.ucla.edu/~asl/bsl/0302-toc.htm> (viewed Jan. 1, 2011).
- [Soare, 1978] R. I. Soare. Recursively enumerable sets and degrees. *The Bulletin of American Mathematical Society*, 84(6):1149–1181, 1978. Available at <http://www.ams.org/bull/1978-84-06/S0002-9904-1978-14552-2> (viewed Dec. 31, 2011).
- [Turing, 1936] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. In *Proceedings of the London Mathematical Society*, volume 42 of *series 2*, pages 230–265, 1936.
- [Turing, 1939] A. M. Turing. Systems of logics based on ordinals. In *Proceedings of the London Mathematical Society*, volume 45 of *series 2*, pages 161–228, 1939. Available at <http://www.turingarchive.org/browse.php/B/15> (viewed Apr. 13, 2011).